A crash course on reinforcement learning

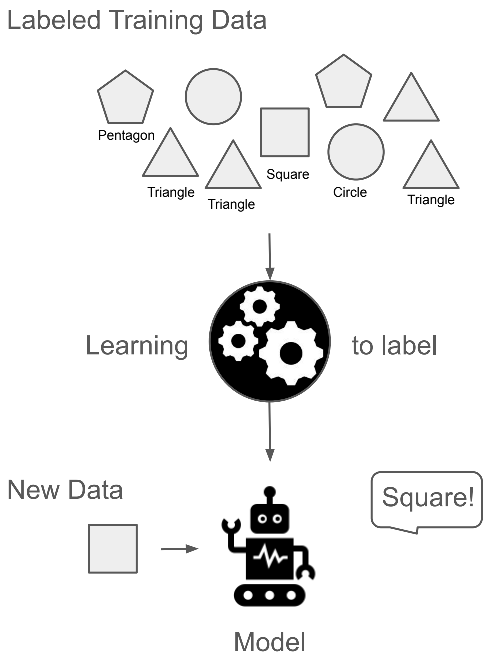
Felix Wagner

Institute of High Energy Physics of the Austrian Academy of Sciences

Inverted CERN School of Computing 2023

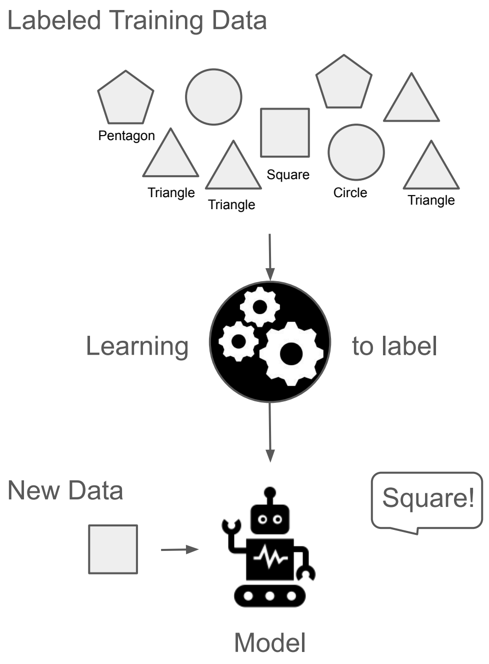
Three types of machine learning

Supervised Learning

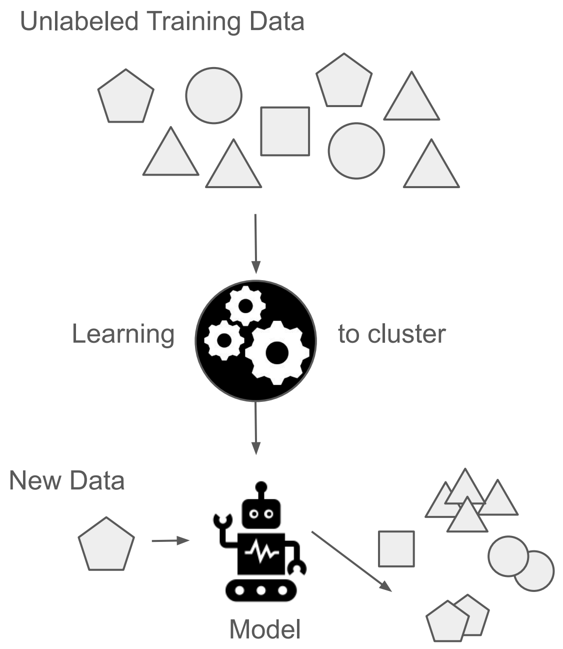


Three types of machine learning

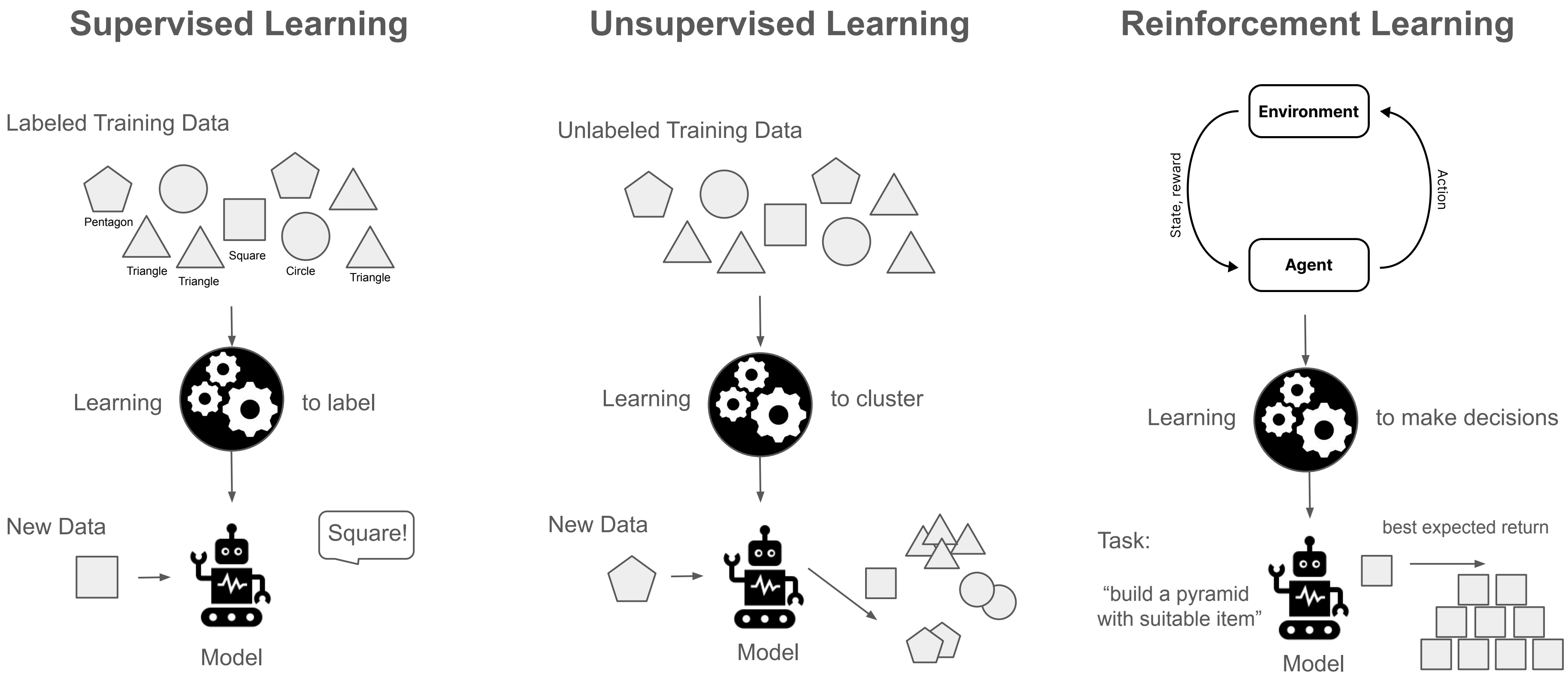
Supervised Learning



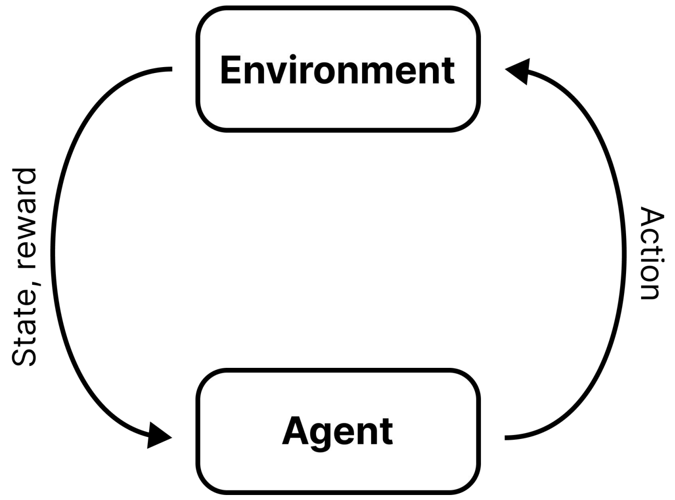
Unsupervised Learning



Three types of machine learning



Reinforcement learning

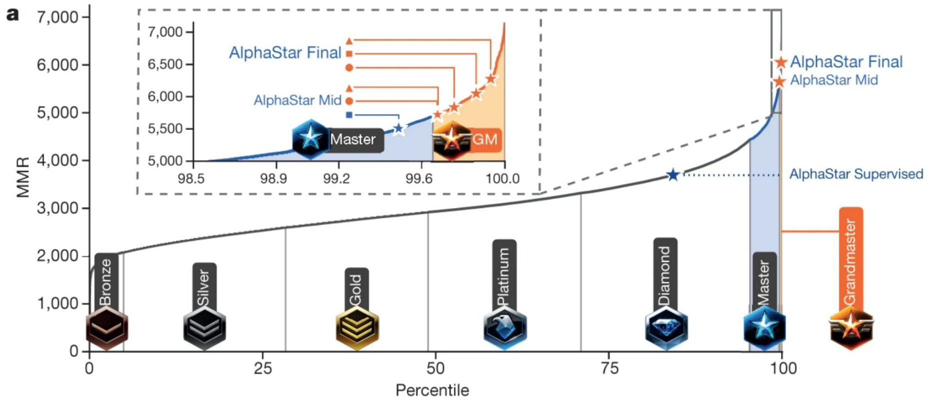


Reinforcement learning (RL) in a nutshell:
“Learn a policy to maximize rewards over time””

Reinforcement learning

There are famous examples of RL for learning games.

"AlphaGo" winning against
the Go world champion



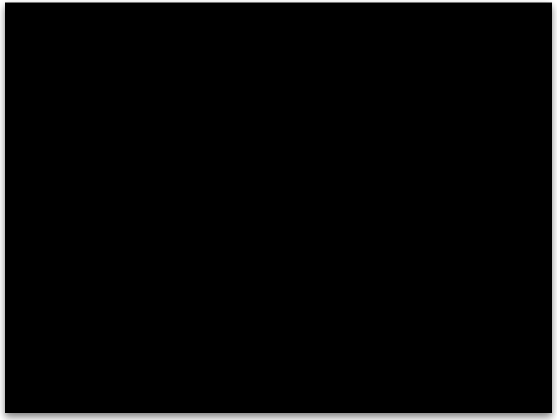
"AlphaStar" wins Starcraft against
99.85% of human players.

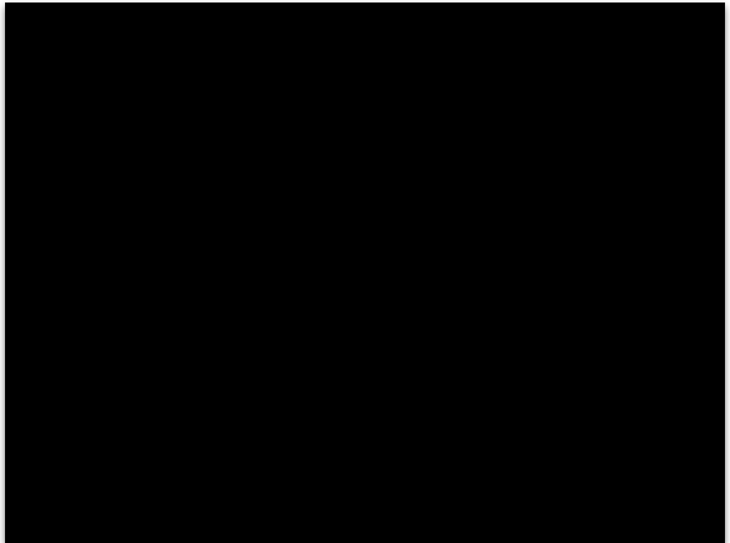
But it can also be applied to real world problems, as …

Reinforcement learning

Videos and GIFs are disabled in the PDF version of these slides.

… a framework for
model-free,
time-discrete control
problems.





OpenAI Gym - A framework for reinforcement learning
https://www.gymlibrary.dev/

This lecture is for you!

Well, if you …

- have ever asked yourself “What would be the **best strategy** to win UNO, chess, black jack, …?”
- work on problems that involve optimizing the control of machines or other types of **goal-oriented action planning**.
- are not frightened by mathematical definitions and linear algebra.
- are generally curious about **machine learning** and **artificial intelligence**.
- and otherwise ready to learn something completely **new and exciting**!

Outline

We cover the following topics:

- *Markov decision processes (MDPs)*
- *Solving small MDPs with tabular methods*
- *Solving large MDPs with policy gradient methods*

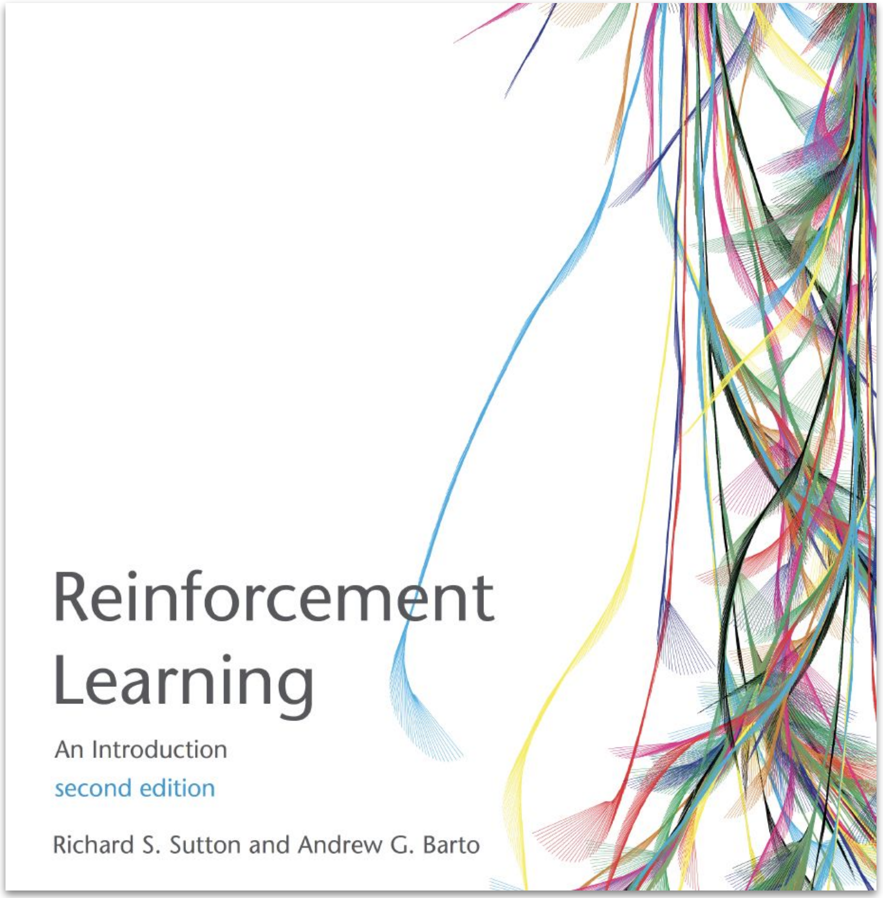
Transfer questions after every section.

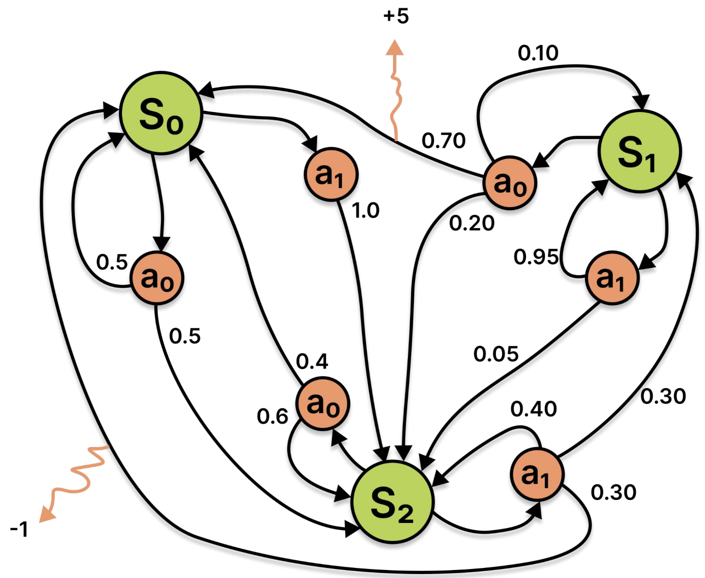
Jupyter notebooks with code examples are hosted on github.com/fewagner/icsc23.

Literature

“Sutton & Barto” is the standard text book for RL - and main source for this lecture.

However, developments in the field are fast and staying up to date involves skimming papers and proceedings all the time.





Markov decision processes (MDPs)

Markov decision processes (MDPs)

An MDP is a 4-tuple consisting of an **action space** and **state space**, a **dynamics function**:

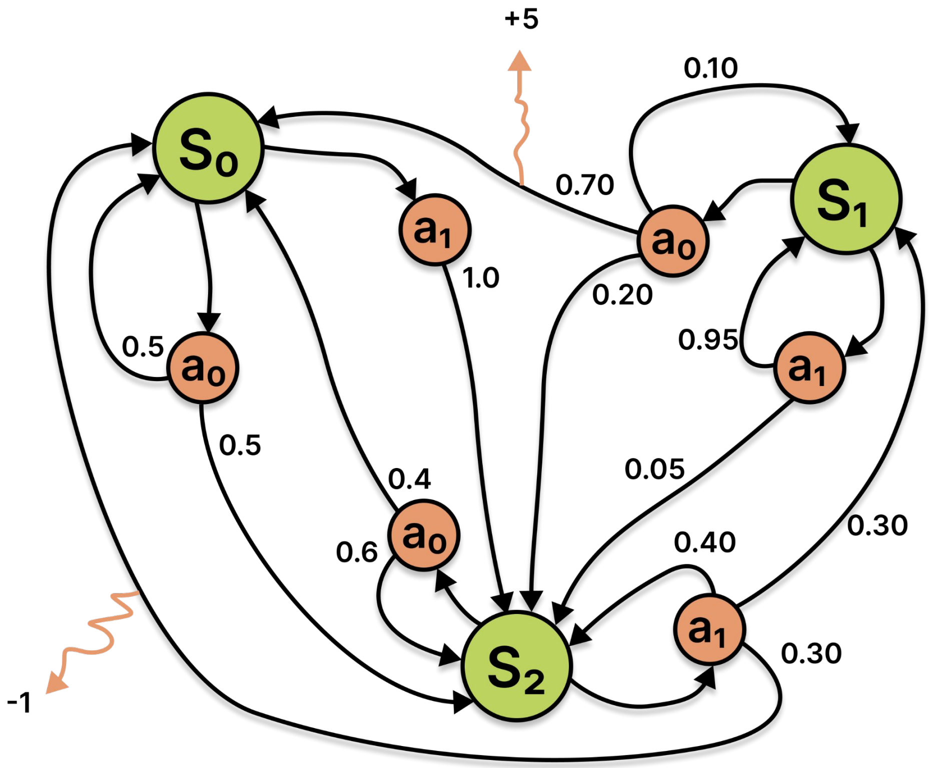
$$p : (A,\, S) \;\mapsto\; \{\text{probabilities for } S'\}$$

and a **reward function**:

$$r : \big(S,\, A,\, S'\big) \;\mapsto\; R \in \mathbb{R}$$

Sometimes this notation is convenient:

$$p\big(S', R \mid S, A\big) \;:=\; \{\text{probabilities for } S' \text{and reward } R\}$$



Markov decision processes (MDPs)

For the MDP displayed on the right:

state space $\mathcal{S} = \{s_0, s_1, s_2\}$

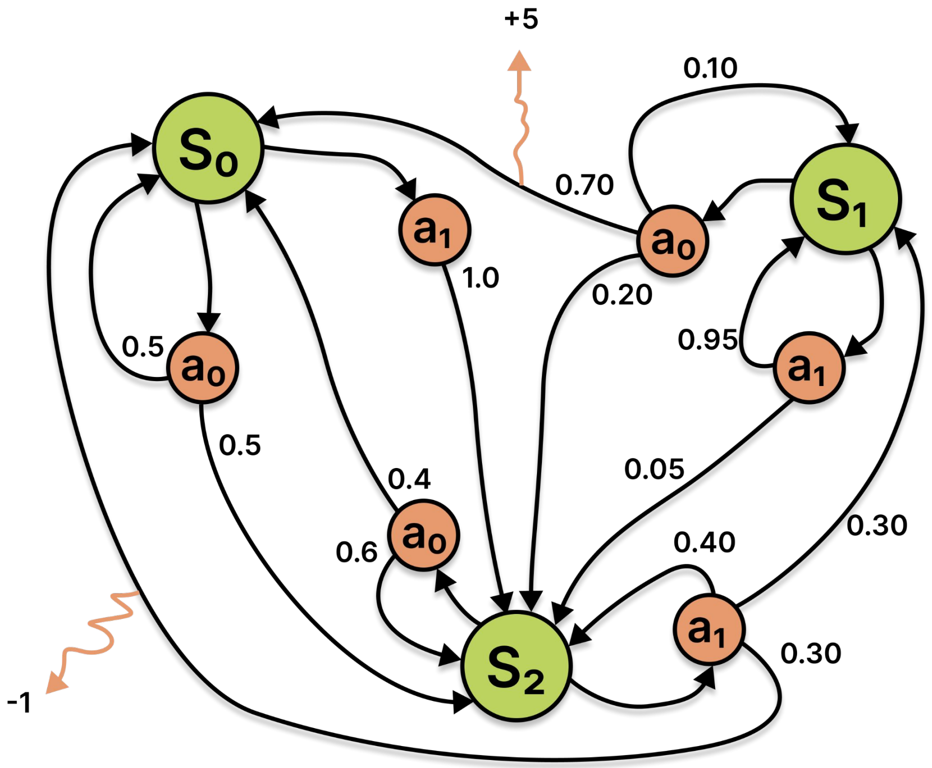
action space $\mathcal{A} = \{a_0, a_1\}$

dynamics function

$$\underbrace{\begin{pmatrix} s_0 \\ s_1 \\ s_2 \end{pmatrix}}_{\text{new state}} \leftarrow \underbrace{\begin{pmatrix} .5 & 0 & .7 & 0 & .4 & .3 \\ 0 & 0 & .1 & .95 & 0 & .3 \\ .5 & 1. & .2 & .05 & .6 & .4 \end{pmatrix}}_{\text{dynamics function}} \underbrace{\begin{pmatrix} (s_0, a_0) \\ (s_0, a_1) \\ (s_1, a_0) \\ (s_1, a_1) \\ (s_2, a_0) \\ (s_2, a_0) \end{pmatrix}}_{\text{state-action pair}}$$

reward function

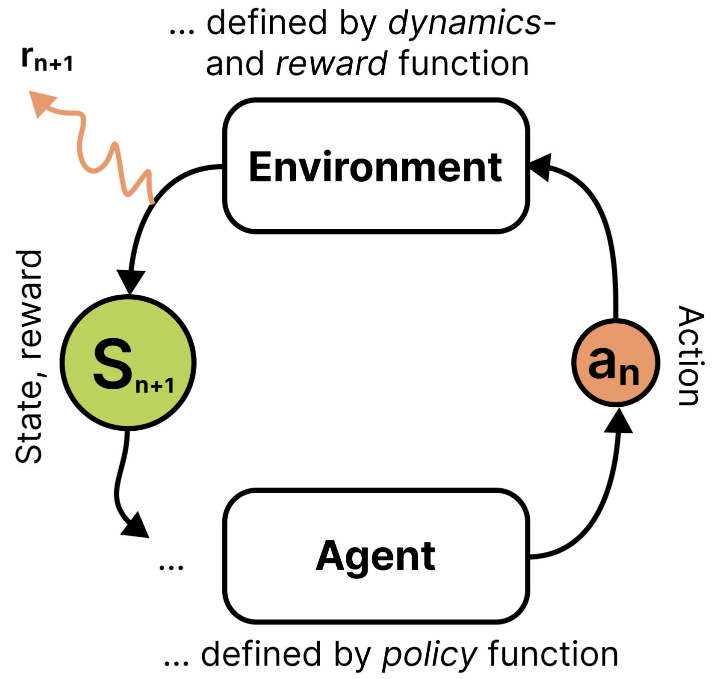
$$r = \begin{cases} +5, & \text{if } (S, A, S') = (s_1, a_0, s_0), \\ -1, & \text{if } (S, A, S') = (s_2, a_1, s_0), \\ 0, & \text{else} \end{cases}$$



Markov decision processes (MDPs)

RL models the interaction of an **agent** with an **environment** (an MDP).

The agent aims to take actions that maximize the collected rewards over time (**returns**), following a **policy** function:



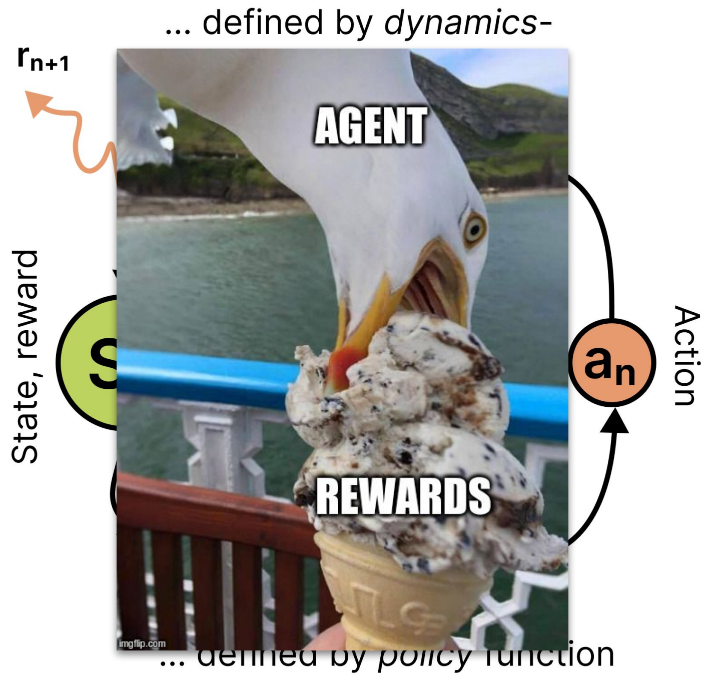
... defined by *dynamics-* and *reward* function
r_{n+1}
Environment
State, reward
S_{n+1}
a_n
Action
...
Agent
... defined by *policy* function

$$\pi : S \mapsto \pi(A \mid S) = \{\text{probabilities for } A \text{ given state } S\}$$

Markov decision processes (MDPs)

RL models the interaction of an **agent** with an **environment** (an MDP).

The agent aims to take actions that maximize the collected rewards over time (**returns**), following a **policy** function:



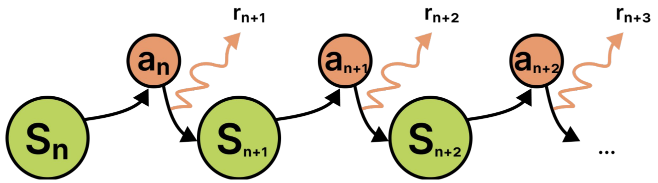
$$\pi : S \mapsto \pi(A \mid S) = \{\text{probabilities for } A \text{ given state } S\}$$

Values and the Bellman equation

For a given policy and MDP, we can calculate the expected returns for any initial state. These are also called the **state values.**

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots$$

$$v_\pi = E_\pi[G_t \mid S_t = s]$$

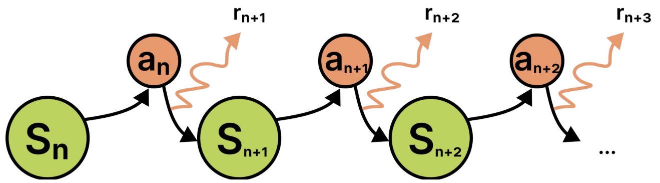


Values and the Bellman equation

For a given policy and MDP, we can calculate the expected returns for any initial state. These are also called the **state values**.

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots$$

$$v_\pi = E_\pi[G_t \mid S_t = s]$$



Some MDPs have terminal states (**episodic** MDPs), others run forever (**continuous** MDPs).

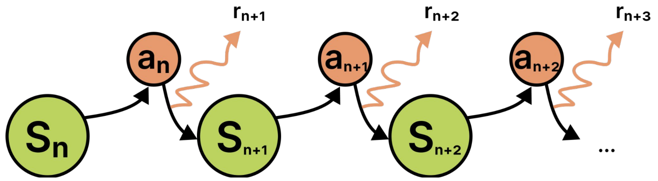
Especially for continuous MDPs values could diverge over time. Therefore we introduce a **discounting factor** 0<γ<1.

Values and the Bellman equation

For a given policy and MDP, we can calculate the expected returns for any initial state. These are also called the **state values.**

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots$$

$$v_\pi = E_\pi[G_t \mid S_t = s]$$



The state values satisfy a recursive relationship that is called the **Bellman equation**:

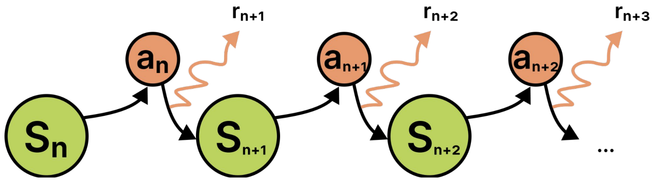
$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

Values and the Bellman equation

For a given policy and MDP, we can calculate the expected returns for any initial state. These are also called the **state values.**

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots$$

$$v_\pi = E_\pi[G_t \mid S_t = s]$$



The state values satisfy a recursive relationship that is called the **Bellman equation**:

Linear system of equations ⇒ values are unique and can be computed with linear algebra.

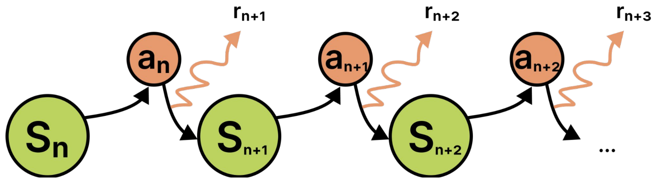
$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

Values and the Bellman equation

For a given policy and MDP, we can calculate the expected returns for any initial state. These are also called the **state values.**

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots$$

$$v_\pi = E_\pi[G_t \mid S_t = s]$$
$$q_\pi = E_\pi[G_t \mid S_t = s, A_t = a]$$



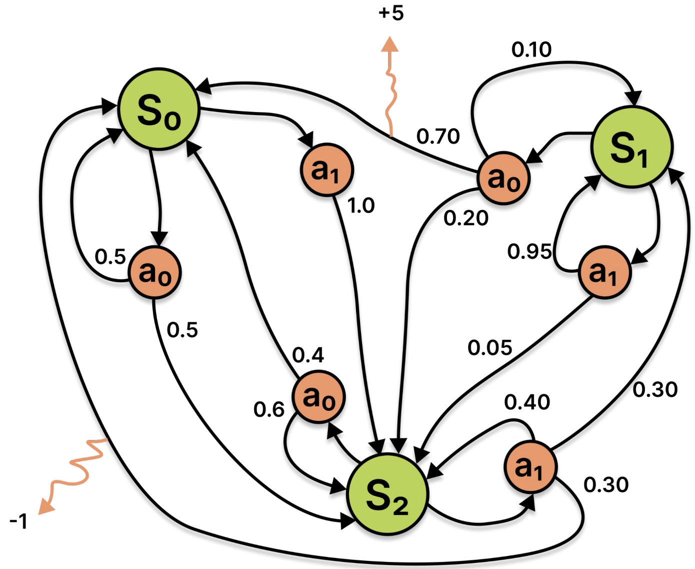
The state values satisfy a recursive relationship that is called the **Bellman equation**:

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

In practice it's often handy to define state-action values instead.

$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

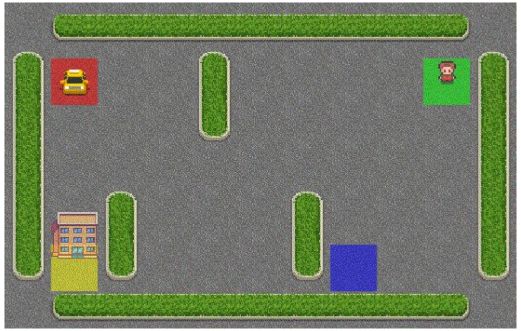
A few numerical experiments with our MDP



Notebook 1 on
github.com/fewagner/icsc23

Some more realistic examples

Taxi

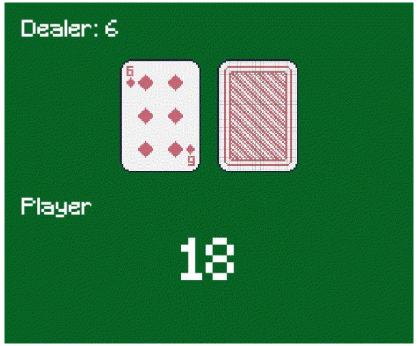


states (500) = {position on 5x4 grid, location and destination passenger g/r/y/b}

actions (6) = {move up/down/left/right, pickup/dropoff passenger}

rewards = +20 for delivering, -10 for wrong dropoff/pickup, -1 else

Black Jack



states (704) = {own points, dealers visible points, whether you hold a usable ace}

actions (2) = {hit, draw}

rewards = +1 win, -1 lose, 0 draw

We solve these environments in section 2!

The Markov property

MDPs are “memoryless”, i.e. the dynamics and reward function do not depend on history prior to the **current state and action**.

$$p : (A,\, S) \mapsto \{\text{probabilities for } S'\}$$

$$r : \left(S,\, A,\, S'\right) \mapsto R \in \mathbb{R}$$

MY NAME'S BILL, AND I'M FORGETFUL.
HI, CARL!
FORGETFUL FOLKS SUPPORT GROUP
4-13
COVERLY ©2016
DIST. BY CREATORS SPEEDBUMP.COM

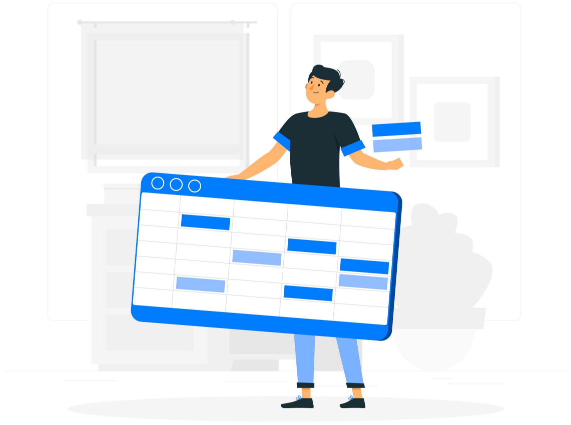
Attention!

Not every RL environment necessarily satisfies the Markov property. Some have unobserved, internal states. These are called partially observable MDPs (**POMDPs**).

MDPs: transfer questions

On which **level of detail** should we formulate the MDP? Consider driving a car. Is the action to "*drive to the shop*", or is it "*turn the turning wheel to the right*", or even "*move the left front wheel by 15 degrees*"?

Are there **limitations** to the requirement of a scalar reward? Can all types of goal-oriented decision making be formulated as maximizing rewards?



Solving small MDPs with tabular methods

A greedy control algorithm

Assume, we know the action-value (Q) function corresponding to an optimal policy.

	a_0	a_1
s_0	1.96	2.1
s_1	5.69	4.72
s_2	2.00	2.54

A greedy control algorithm

Assume, we know the action-value (Q) function corresponding to an optimal policy.

We can just always take the action with the highest Q value!

	a_0	a_1
s_0	1.96	2.1
s_1	**5.69**	4.72
s_2	2.00	2.54

In practice, it’s not that easy …

Let’s look back at the Bellman equation

$$q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

We need to learn the values from data

The expected value depends on the dynamics of the environment and has to be learned from experience.

$$q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

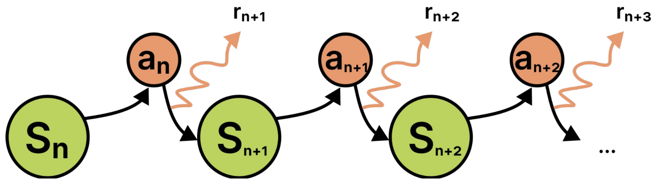
We need to sample data cleverly

The expected value depends on the dynamics of the environment and has to be learned from experience.

$$q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

An efficient algorithm collects preferably experience that is relevant for finding an optimal policy.

Temporal difference (TD) learning

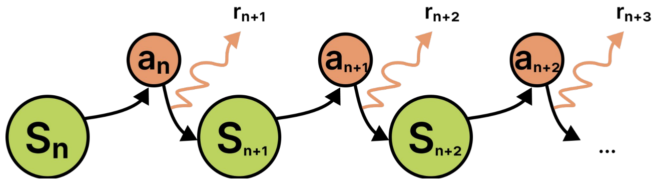


"If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be TD learning."
- Sutton & Barto, first sentence of the TD chapter

General idea of TD learning:

- Implement a policy based on the Q-values to choose next action.
- After every step, make update on Q-values.

Temporal difference (TD) learning



"If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be TD learning."
- Sutton & Barto, first sentence of the TD chapter

General idea of TD learning:

- **Implement a policy based on the Q-values to choose next action.**
- After every step, make update on Q-values.

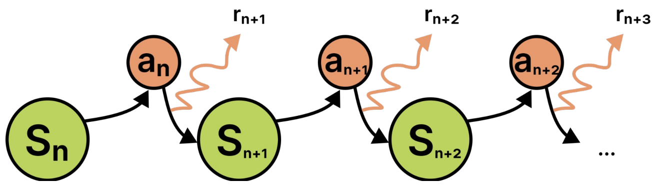
Exploration vs. exploitation

Rewards in RL are typically sparse and need to be discovered before they can be propagated to neighboring and distant state values.

“Epsilon-greedy” policy:

Take greedy action with probability 1-ε and random action with probability ε.

Temporal difference (TD) learning



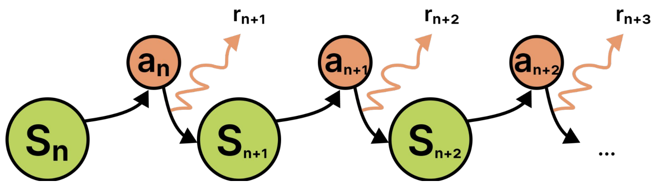
"If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be TD learning."

- Sutton & Barto, first sentence of the TD chapter

General idea of TD learning:

- **"Epsilon-greedy" policy**
- After every step, make update on Q-values.

Temporal difference (TD) learning



"If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be TD learning."

- Sutton & Barto, first sentence of the TD chapter

General idea of TD learning:

- "Epsilon-greedy" policy

- **After every step, make update on Q-values.**

Let’s look back at the Bellman equation

Can’t we learn an expected value iteratively?

$$q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

We can use the Bellman equation as an update rule

$$Q(S_{t,} A_t) \leftarrow Q(S_t, A_t) + \underset{\uparrow}{\alpha} \underbrace{[R_{t+1} + \gamma Q(S_{t+1}, a) - Q(S_t, A_t)]}_{\delta_t \text{ ... TD error}}$$

learning rate ∈ (0,1)

We can use the Bellman equation as an update rule

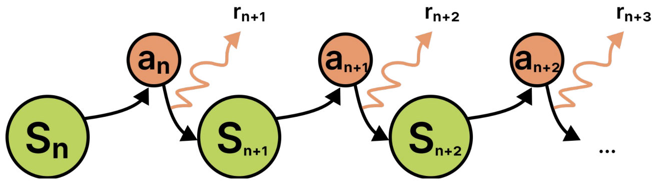
$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \underbrace{[R_{t+1} + \gamma Q(S_{t+1}, a) - Q(S_t, A_t)]}_{\delta_t \text{ ... TD error}}$$

learning rate ∊ (0,1)

We are moving our *previous estimate* of the Q-value a small step towards the *new experience*!

Temporal difference (TD) learning



"If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be TD learning."
- Sutton & Barto, first sentence of the TD chapter

General idea of TD learning:

- **"Epsilon-greedy" policy**

- **Update rule:** $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \underbrace{[R_{t+1} + \gamma Q(S_{t+1}, a) - Q(S_t, A_t)]}_{\delta_t \text{ ... TD error}}$

On-policy and off-policy methods

The simplest on/off policy control schemes :

SARSA $$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Q-learning $$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

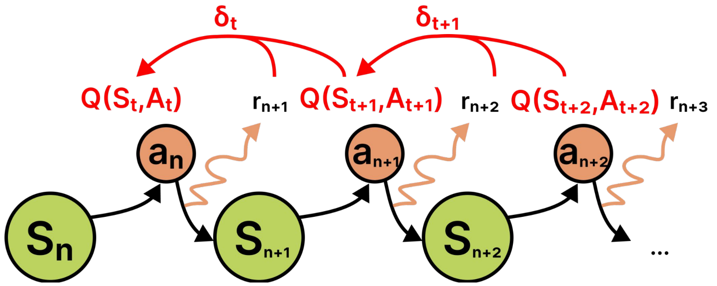
On-policy: SARSA considers that future actions are taken according to current policy.

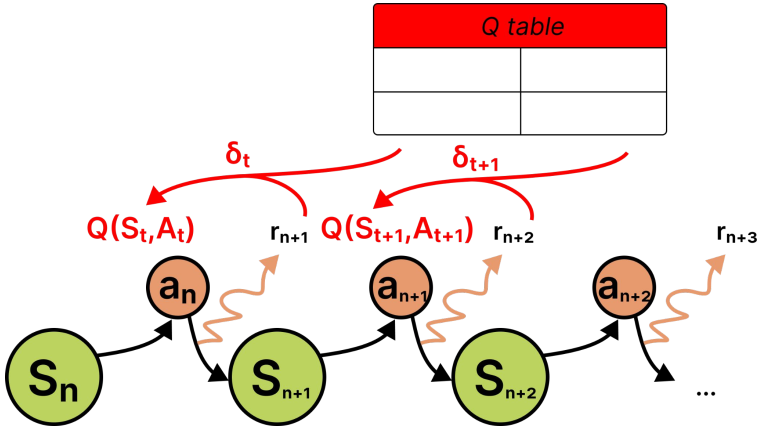
Off-policy: Q-learning considers that future actions are taken with another (target) policy, in this case the greedy policy.

can learn policy without actually following it!

On-policy and off-policy methods

SARSA $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$



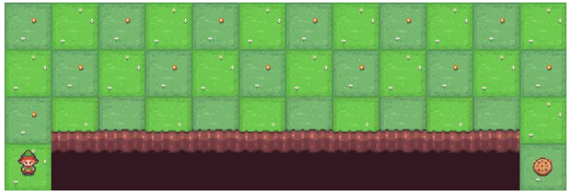


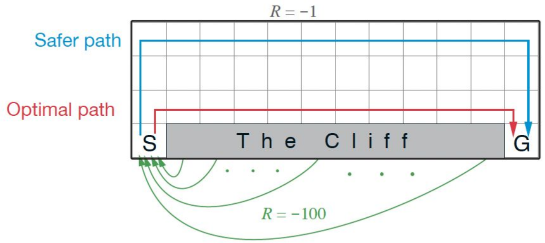
Q-learning $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$

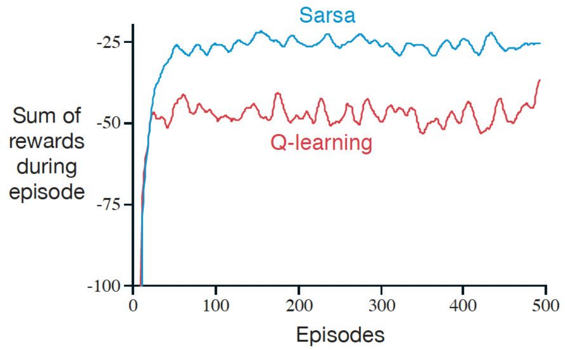
Cliff walking

The agents walks in a gridworld, receiving -1 until it reaches the goal, and -100 for falling off the cliff.

Q-learning **learns the optimal policy** to walk next to the cliff, but falls off sometimes due to the ε-greedy action selection. SARSA considers this action selection and obtains **higher rewards online**.



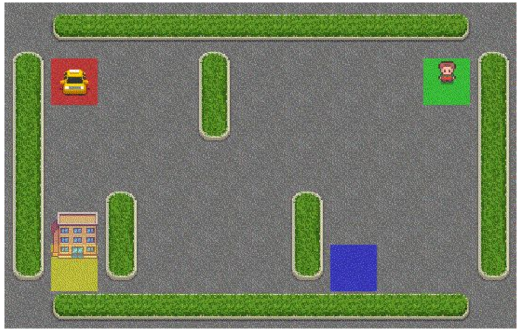


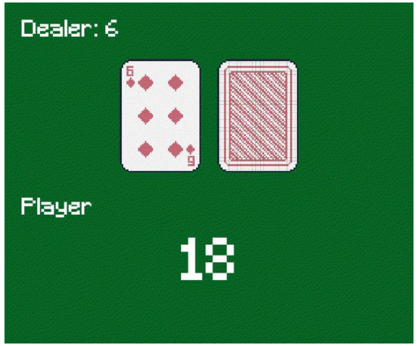


(from Sutton & Barto)

SARSA/Q-learning on taxi driver/black jack

Notebook 3 on
github.com/fewagner/icsc23

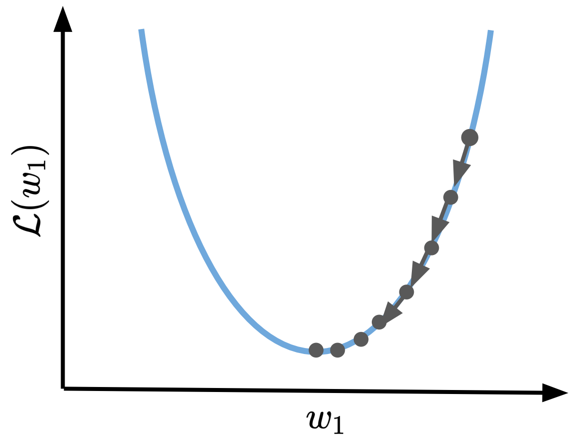




Tabular methods: transfer questions

What problems could occur with the **epsilon-greedy** scheme?

When would you use an **on-policy** method (SARSA) and when an **off-policy** method (Q-learning)?



Solving large MDPs with policy gradient methods

How large is a large MDP?

- How many states has the game **tic-tac-toe**?

How large is a large MDP?

- How many states has the game **tic-tac-toe**? 765
- How many states has the game **go**?

How large is a large MDP?

- How many states has the game **tic-tac-toe**? 765
- How many states has the game **go**? $3^{361}=10^{172}$
- How many states has **driving a car**?

for comparison: our universe has ~10^{82} atoms

How large is a large MDP?

- How many states has the game **tic-tac-toe**? 765
- How many states has the game **go**? $3^{361} = 10^{172}$
- How many states has **driving a car**? (all positions and velocities of wheels, all sensor readings, visual input, GPS data, … ???)

How would we set up a Q-table for this?

Function approximation

Many environment have *continuous* (real numbers) state and action spaces. We cannot build and exact Q-table for such environments!

Instead, we treat the Q-table as a function, called **value function**:

$$\hat{q}_w(s, a) \in \mathbb{R}$$

In this formulation, we can use **function approximators** to learn the value function, similarly as we updated the Q table.

But … *how to choose a greedy action for continuous action spaces*? 🤔

Function approximation

Many environment have *continuous* (real numbers) state and action spaces. We cannot build and exact Q-table for such environments!

Instead, we treat the Q-table as a function, called **value function**:

$$\hat{q}_w(s, a) \in \mathbb{R}$$

In this formulation, we can use **function approximators** to learn the value function, similarly as we updated the Q table.

For large or continuous action spaces, we can treat the policy as a **policy function**:

$$\pi_\theta(a|s) \in [0, 1],\ \pi_\theta(\cdot|s) \ldots \text{probability distribution}$$

We call this value a “preference”.

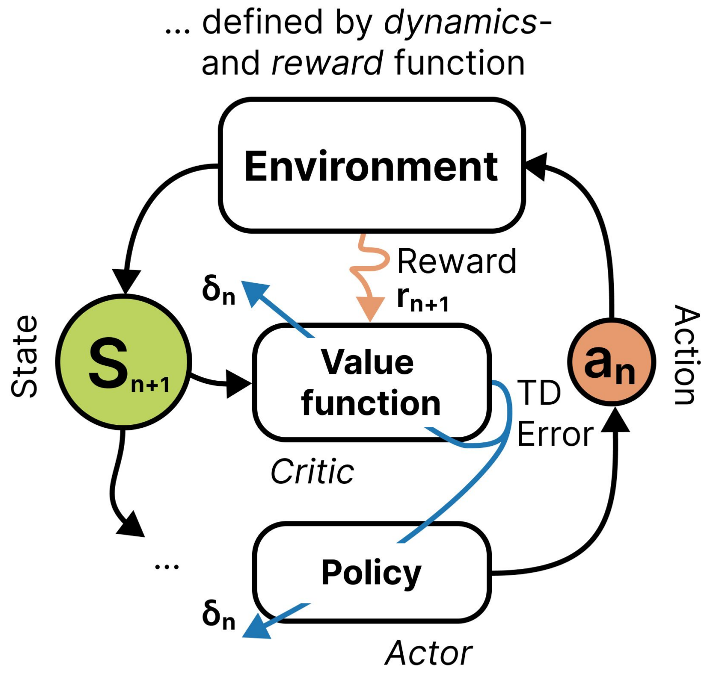
There are many ways to approximate a function, we focus on parametric approximators.
w and Θ are the real-values parameter vectors.

Actor-critic: TD learning in continuous spaces

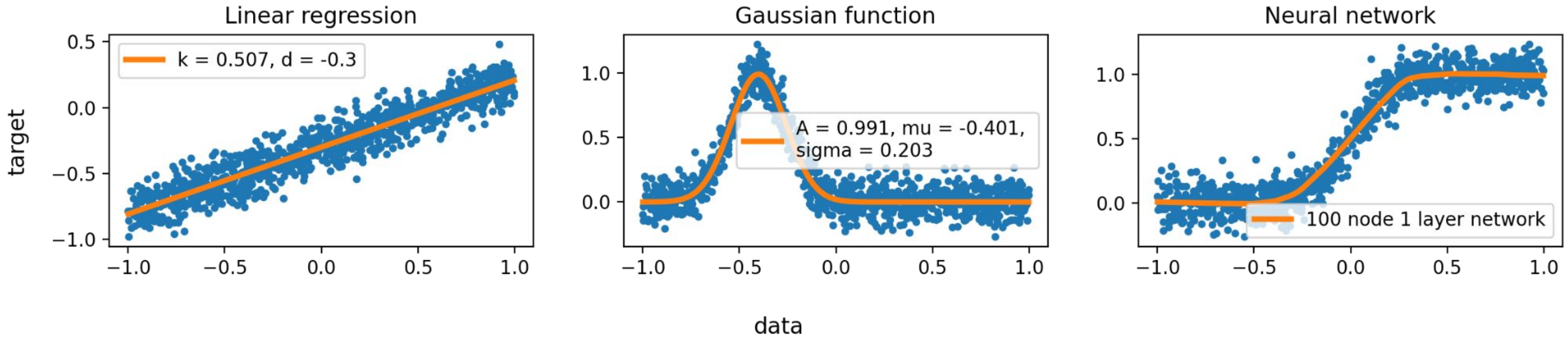
Simplest version of an **actor-critic** agent:

The value function (**critic**) estimates the returns for states.

The policy (**actor**) learns actions that have high values.



Examples of parametric function approximators



linear regression

Gaussian radial basis function

neural network

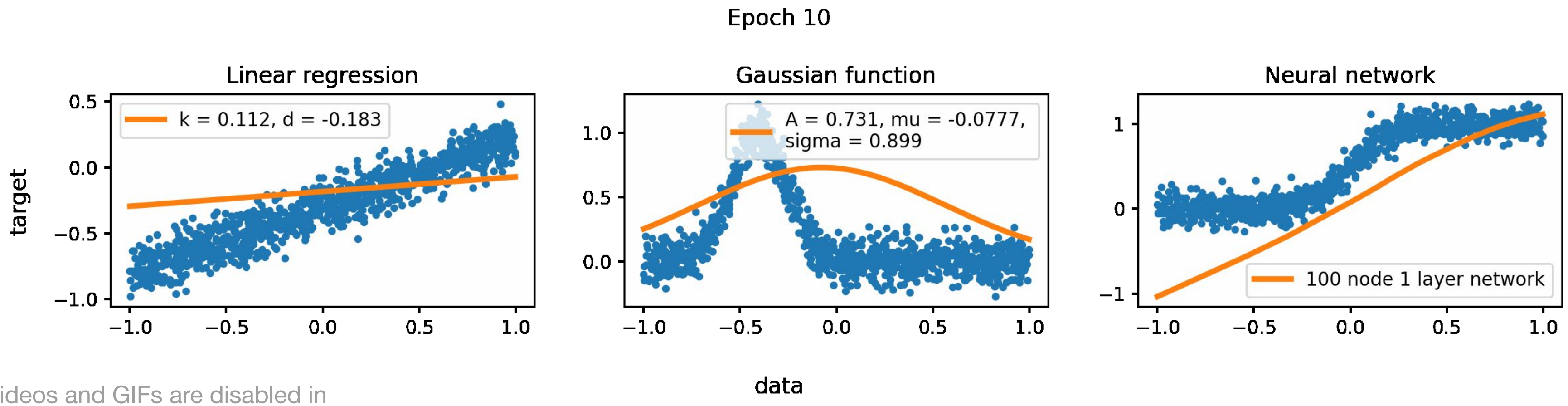
$$y_{k,d}(x) = kx + d$$

$$y_{a,\mu,\sigma}(x) = a \exp\left(\frac{(x-\mu)^2}{2\sigma^2}\right)$$

$$y_{\vec{w_0},\vec{b_0},\vec{w_1},b_1}(x) = \vec{w_1}\sigma\left(\vec{w_0}x + \vec{b_0}\right) + b_1$$

Learning parameters with gradient descent

$$\boxed{w_{t+1} = w_t + \alpha \nabla \mathcal{L}(w_t)}$$



Videos and GIFs are disabled in the PDF version of these slides.

Notebook 4 on github.com/fewagner/icsc23

Loss functions for values/policy

Value function loss: minimize *mean squared error* between returns and values, leading to gradient update

$$w_{t+1} = w_t + \alpha_w \left(R_{t+1} + \gamma \hat{v}_w(S_{t+1}) - v_w(\hat{S}_t) \right) \nabla_w \hat{v_{w_t}}(S_t)$$

Policy function loss: maximize *probability for actions with high TD error*, leading to gradient update

$$\theta_{t+1} = \theta_t + \alpha_\theta \gamma^t \left(R_{t+1} + \gamma \hat{v}_w(S_{t+1}) - v_w(\hat{S}_t) \right) \nabla_\theta \ln \pi_{\theta_t}(A_t | S_t)$$

Actor-critic: TD learning in continuous spaces

Simplest version of an **actor-critic** agent:

The value function (**critic**) estimates the returns for states.

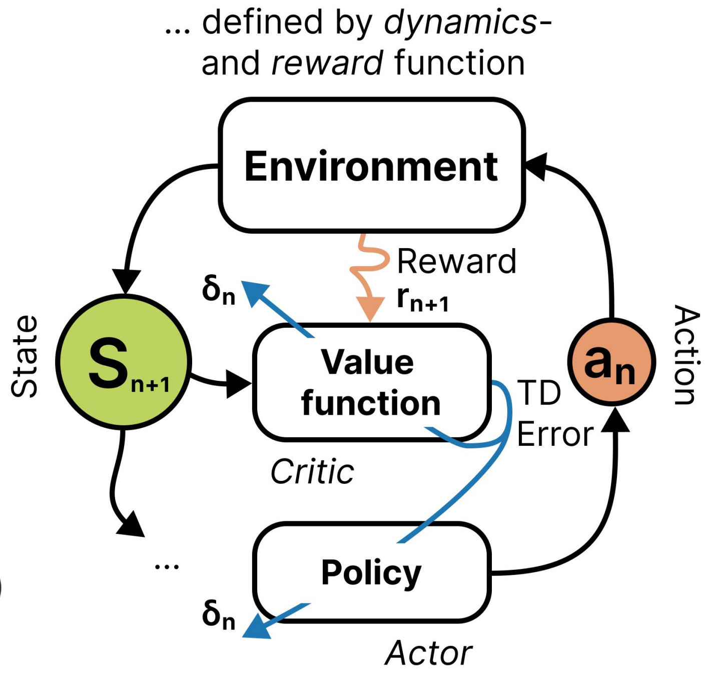
The policy (**actor**) learns actions that have high values.

Exploration-exploitation is balanced by **sampling actions** from the policy's probability distribution.

After every step, updates are done according to:

$$w_{t+1} = w_t + \alpha_w \Big(R_{t+1} + \gamma \hat{v}_w(S_{t+1}) - v_w(\hat{S}_t)\Big) \nabla_w \hat{v_{w_t}}(S_t)$$
$$\theta_{t+1} = \theta_t + \alpha_\theta \gamma^t \Big(R_{t+1} + \gamma \hat{v}_w(S_{t+1}) - v_w(\hat{S}_t)\Big) \nabla_\theta \ln \pi_{\theta_t}(A_t|S_t)$$

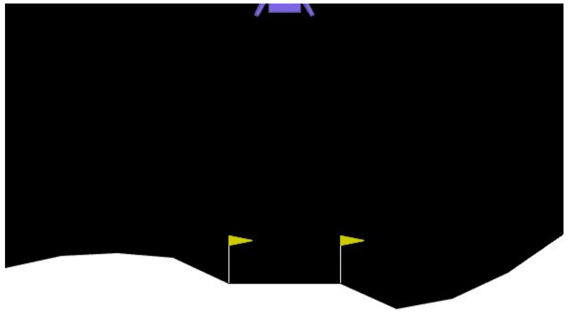
In this formulation, the algorithm can be only used for **episodic tasks** and uses a **state value function**.



Actor-critic on the lunar lander

Notebook 5 on
github.com/fewagner/icsc23

Videos and GIFs are disabled in the PDF version of these slides.



OpenAI Gym:
LunarLander-v2

Policy gradient methods: transfer questions

When would you prefer a method that uses **approximation over a tabular** method?

Is our version of actor critic an **on-policy** or **off-policy** algorithm?

Actor-critic (?)

$$w_{t+1} = w_t + \alpha_w \Big(R_{t+1} + \gamma \hat{v}_w(S_{t+1}) - v_w(\hat{S}_t) \Big) \nabla_w \hat{v_{w_t}}(S_t)$$

$$\theta_{t+1} = \theta_t + \alpha_\theta \gamma^t \Big(R_{t+1} + \gamma \hat{v}_w(S_{t+1}) - v_w(\hat{S}_t) \Big) \nabla_\theta \ln \pi_{\theta_t}(A_t|S_t)$$

SARSA (on-policy)

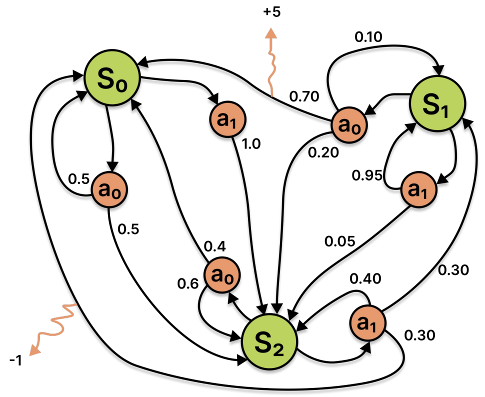
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, \mathbf{A_{t+1}}) - Q(S_t, A_t)]$$

Q-learning (off-policy)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_{\mathbf{a}} Q(S_{t+1}, \mathbf{a}) - Q(S_t, A_t) \Big]$$

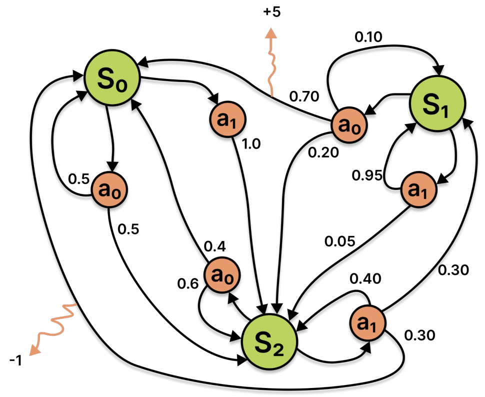
Recap

Recap

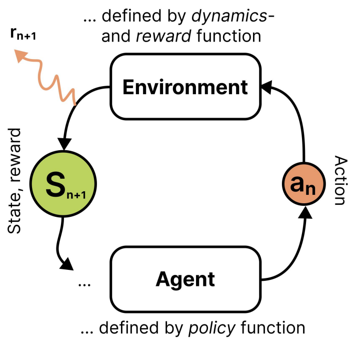


Markov decision processes

Recap



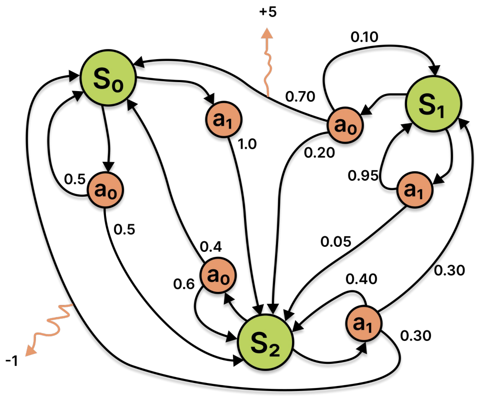
Markov decision processes



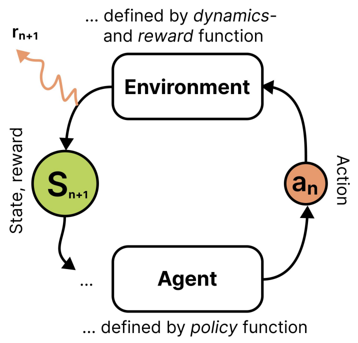
... defined by *dynamics*- and *reward* function
r n+1
Environment
State, reward
S n+1
a n
Action
...
Agent
... defined by *policy* function

Policy, dynamics- and reward function

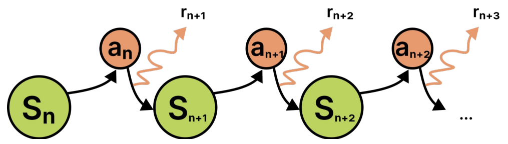
Recap



Markov decision processes


... defined by *dynamics*- and *reward* function
r_{n+1}
Environment
State, reward
S_{n+1}
a_n
Action
...
Agent
... defined by *policy* function

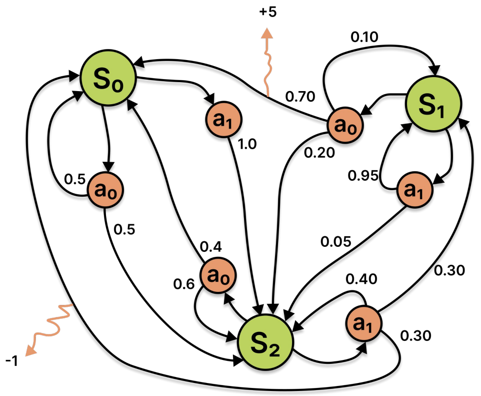
Policy, dynamics- and reward function



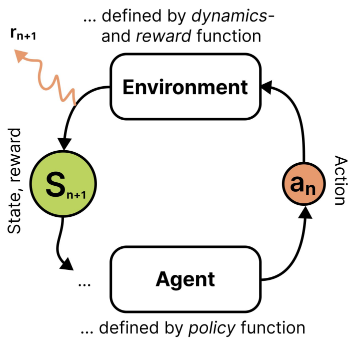
$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

Values, Bellman equation

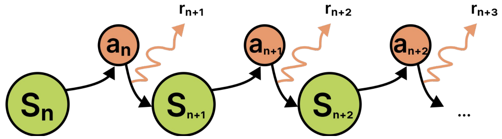
Recap



Markov decision processes

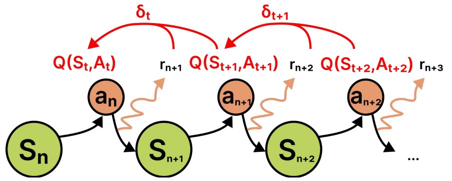

... defined by *dynamics-* and *reward* function
rn+1
Environment
State, reward
Sn+1
an
Action
...
Agent
... defined by *policy* function

Policy, dynamics- and reward function



$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

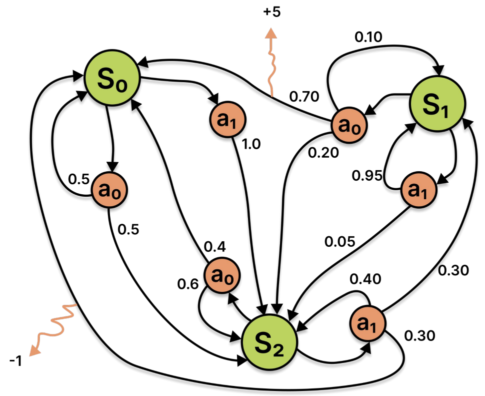
Values, Bellman equation



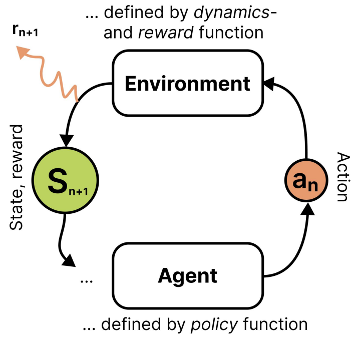
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

SARSA (on-policy)

Recap

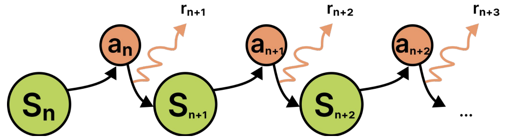


Markov decision processes



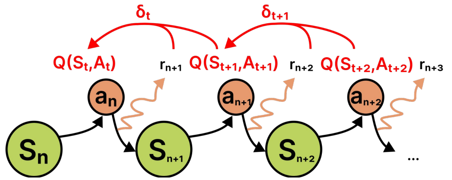
... defined by *dynamics*- and *reward* function
r_{n+1}
Environment
State, reward
S_{n+1}
a_n
Action
...
Agent
... defined by *policy* function

Policy, dynamics- and reward function



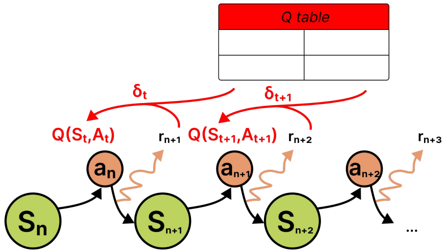
$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

Values, Bellman equation



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

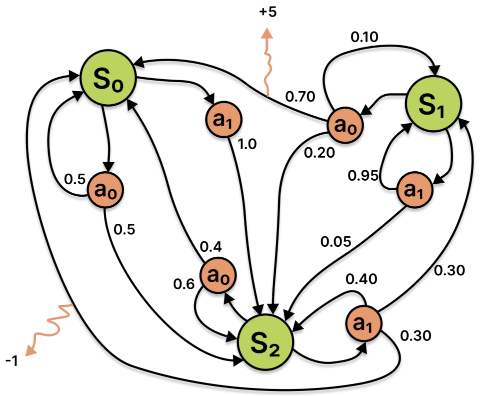
SARSA (on-policy)



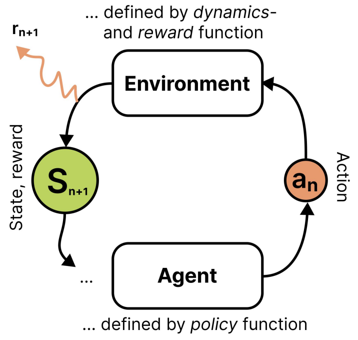
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

Q-learning (off-policy)

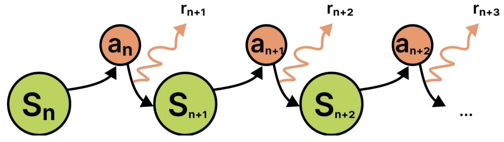
Recap



Markov decision processes

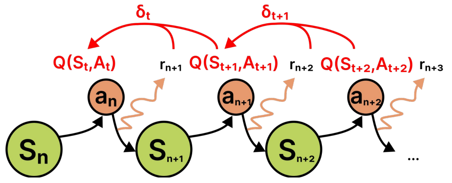

... defined by *dynamics*- and *reward* function
rn+1
Environment
State, reward
Sn+1
an
Action
Agent
...
... defined by *policy* function

Policy, dynamics- and reward function



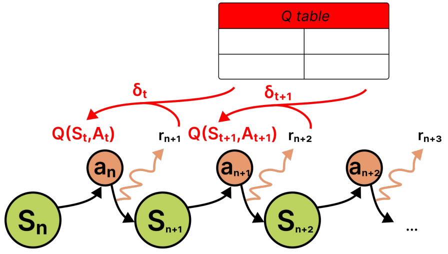
$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

Values, Bellman equation



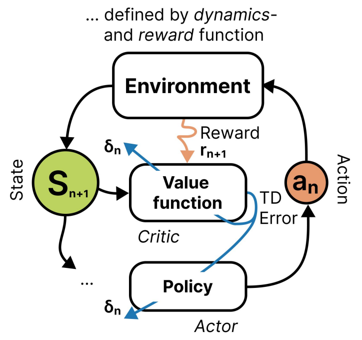
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

SARSA (on-policy)



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

Q-learning (off-policy)



Actor-critic

model-free/model-based

partially observable MDPs

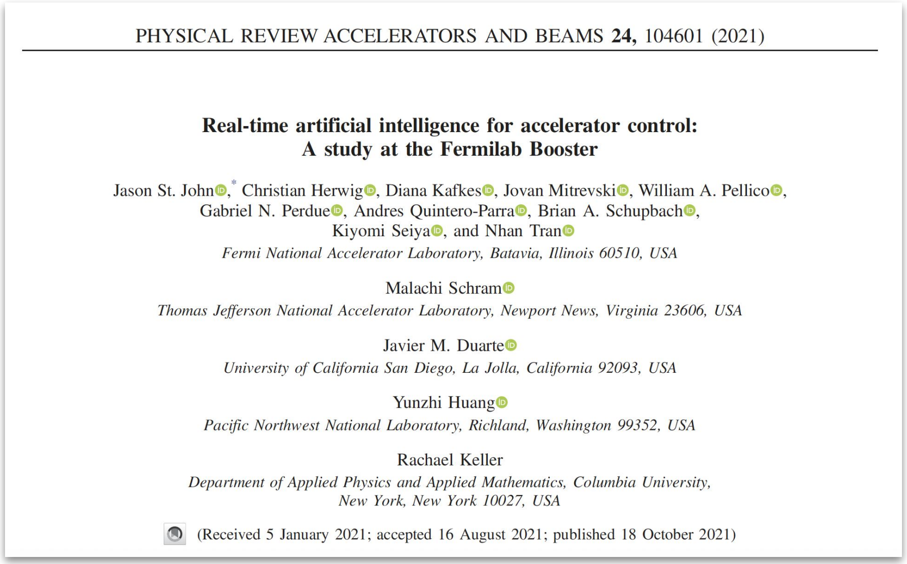
prediction/control

bandits

There is much more to learn …

A few examples of reinforcement learning in physics.

Reinforcement learning in (experimental) physics



PHYSICAL REVIEW ACCELERATORS AND BEAMS **24,** 104601 (2021)

Real-time artificial intelligence for accelerator control: A study at the Fermilab Booster

Jason St. John,* Christian Herwig, Diana Kafkes, Jovan Mitrevski, William A. Pellico, Gabriel N. Perdue, Andres Quintero-Parra, Brian A. Schupbach, Kiyomi Seiya, and Nhan Tran

Fermi National Accelerator Laboratory, Batavia, Illinois 60510, USA

Malachi Schram

Thomas Jefferson National Accelerator Laboratory, Newport News, Virginia 23606, USA

Javier M. Duarte

University of California San Diego, La Jolla, California 92093, USA

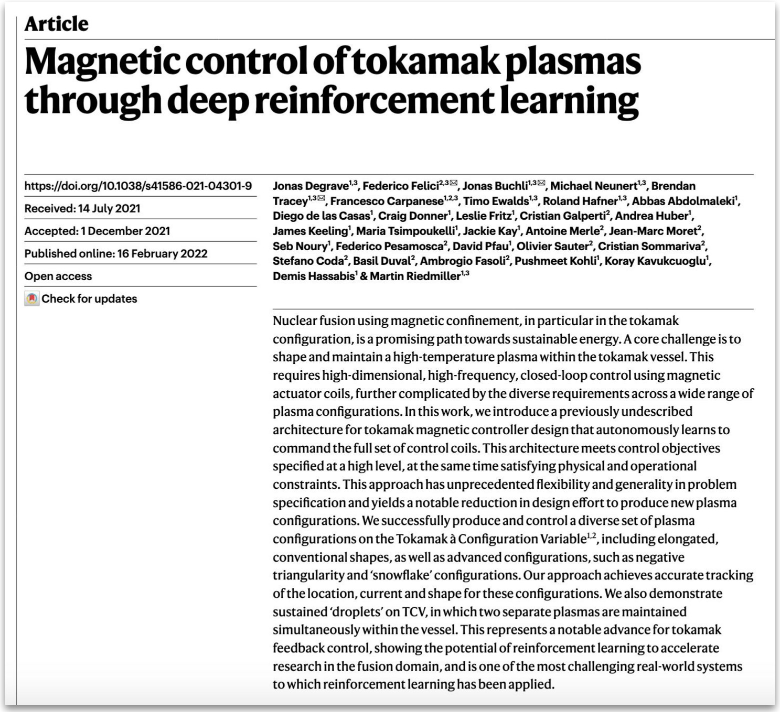
Yunzhi Huang

Pacific Northwest National Laboratory, Richland, Washington 99352, USA

Rachael Keller

Department of Applied Physics and Applied Mathematics, Columbia University, New York, New York 10027, USA

(Received 5 January 2021; accepted 16 August 2021; published 18 October 2021)



Article

Magnetic control of tokamak plasmas through deep reinforcement learning

https://doi.org/10.1038/s41586-021-04301-9

Received: 14 July 2021

Accepted: 1 December 2021

Published online: 16 February 2022

Open access

Check for updates

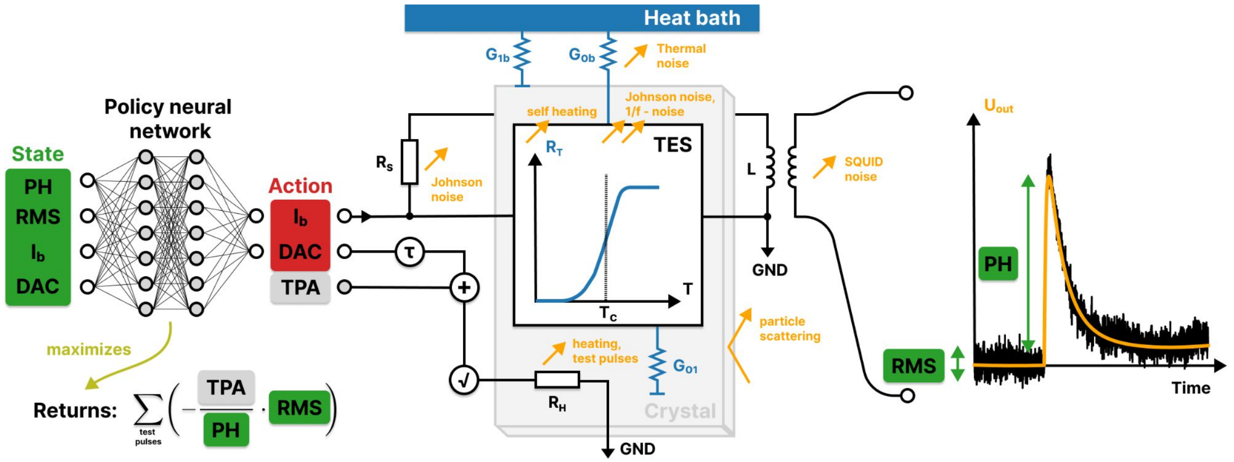
Jonas Degrave[1,3], Federico Felici[2,3✉], Jonas Buchli[1,3✉], Michael Neunert[1,3], Brendan Tracey[1,3✉], Francesco Carpanese[1,2,3], Timo Ewalds[1,3], Roland Hafner[1,3], Abbas Abdolmaleki[1], Diego de las Casas[1], Craig Donner[1], Leslie Fritz[1], Cristian Galperti[2], Andrea Huber[1], James Keeling[1], Maria Tsimpoukelli[1], Jackie Kay[1], Antoine Merle[2], Jean-Marc Moret[2], Seb Noury[1], Federico Pesamosca[2], David Pfau[1], Olivier Sauter[2], Cristian Sommariva[2], Stefano Coda[2], Basil Duval[2], Ambrogio Fasoli[2], Pushmeet Kohli[1], Koray Kavukcuoglu[1], Demis Hassabis[1] & Martin Riedmiller[1,3]

Nuclear fusion using magnetic confinement, in particular in the tokamak configuration, is a promising path towards sustainable energy. A core challenge is to shape and maintain a high-temperature plasma within the tokamak vessel. This requires high-dimensional, high-frequency, closed-loop control using magnetic actuator coils, further complicated by the diverse requirements across a wide range of plasma configurations. In this work, we introduce a previously undescribed architecture for tokamak magnetic controller design that autonomously learns to command the full set of control coils. This architecture meets control objectives specified at a high level, at the same time satisfying physical and operational constraints. This approach has unprecedented flexibility and generality in problem specification and yields a notable reduction in design effort to produce new plasma configurations. We successfully produce and control a diverse set of plasma configurations on the Tokamak à Configuration Variable[1,2], including elongated, conventional shapes, as well as advanced configurations, such as negative triangularity and 'snowflake' configurations. Our approach achieves accurate tracking of the location, current and shape for these configurations. We also demonstrate sustained 'droplets' on TCV, in which two separate plasmas are maintained simultaneously within the vessel. This represents a notable advance for tokamak feedback control, showing the potential of reinforcement learning to accelerate research in the fusion domain, and is one of the most challenging real-world systems to which reinforcement learning has been applied.

Phys. Rev. Accel. Beams **24**, 104601 - 104618 (2021).
https://doi.org/10.1103/PhysRevAccelBeams.24.104601

Nature **602**, 414–419 (2022).
https://doi.org/10.1038/s41586-021-04301-9

Reinforcement learning in (experimental) physics



Optimal operation of cryogenic calorimeters with reinforcement learning
(*paper in preparation*)

Questions?

Backup

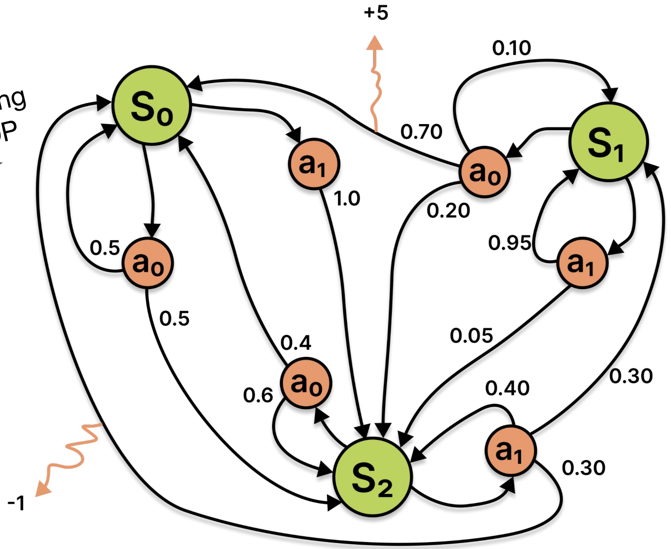
Bandits - contextual bandits - reinforcement learning

Bandits consider only immediate rewards.

Bandit: "The actions bring on average:

a_0	a_1
1.2	-0.1

considering this MDP



Contextual bandit: "The action-state pairs bring on average the rewards:

S_0,a_0	S_0,a_1	S_1,a_0	S_1,a_1	S_2,a_0	S_2,a_1
0	0	3.5	0	0	-0.3

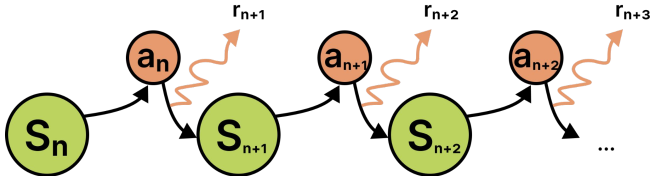
RL: "We have to plan action-state trajectories ahead and consider **delayed rewards**! 🤓 But …how to assign credit to individual action…? 🤔"

We discuss this in section 2 and 3!

Derivation of the Bellman equation

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots$$

$$v_\pi = E_\pi[G_t \mid S_t = s]$$



$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \Big[r + \gamma \mathbb{E}\big[G_{t+1} \mid S_{t+1} = s'\big]\Big] \\ &= \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) \big[r + \gamma v_\pi(s') \big] \end{aligned}$$

The tiger problem, a POMDP

cute, but
dangerous

valuable, but not
worth dying for

For POMDPs, the agent obtains observations of the environment, while the state is a **hidden internal property** of the environment.

states = {tiger right, tiger left}

actions = {open left, open right, listen}

observations = {hear tiger right, hear tiger left}, 85% probability to hear tiger on correct side

rewards = 10 for opening door with treasure, -100 for opening door with tiger, -1 for listening

Prediction and control

“Solve” an MDP can mean two separate thing: solving a prediction and a control problem. For many algorithms (especially for control) the problems are solved simultaneously and iteratively.

Prediction: policy is fixed, predict the returns for given starting states

Control: find a policy that maximizes the returns

(does not necessarily need values)

Model-based and model-free

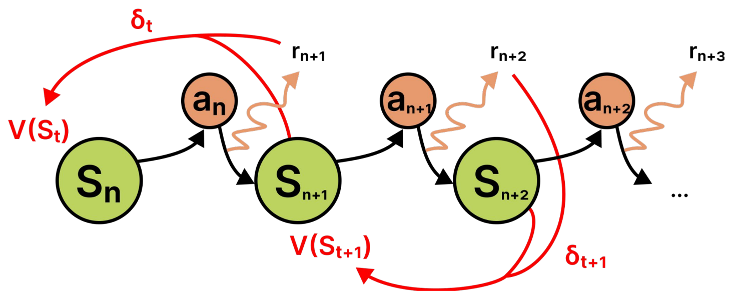
If we only knew everything … 🤷

In general we do not know the dynamics- and reward function (**model-free**)!

For problems with known environment we can use **model-based** techniques: planing, ...

Temporal difference (TD) prediction

“If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be TD learning.”
- Sutton & Barto, first sentence of the TD chapter



General idea of TD prediction:

Fix a policy.

- Let an agent take actions in an environment according to policy.
- After every step, make update on values according to Bellman equation:

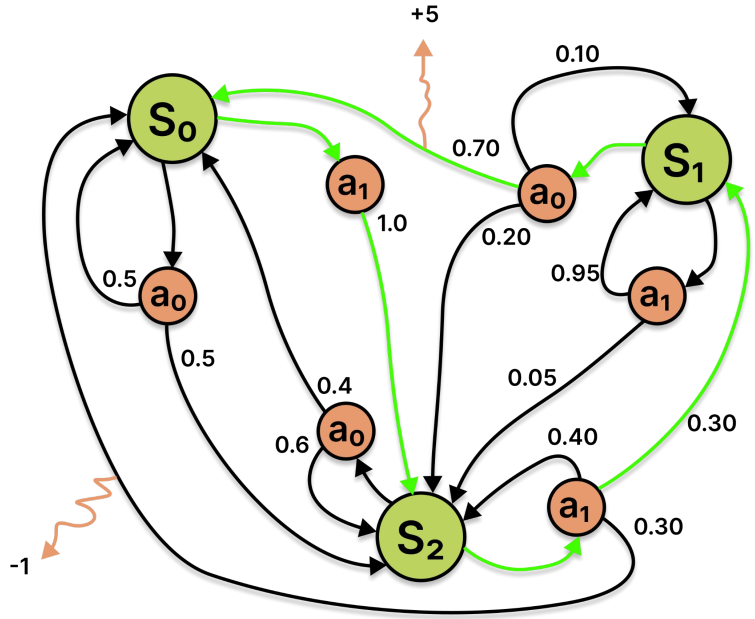
$$V(S_t) \leftarrow V(S_t) + \alpha \underbrace{[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]}_{\delta_t \text{ ... TD error}}$$

learning rate ∈ (0,1)

Scheme with this update rule is also called TD(0)

TD prediction on our MDP

Notebook 2 on
github.com/fewagner/icsc23



Approximating values with gradient descent

$$\boxed{w_{t+1} = w_t + \alpha \nabla \mathcal{L}(w_t)}$$

We choose a function approximator for our value function and update its parameters such that the squared errors with the true value function are minimized:

$$\begin{aligned}
w_{t+1} &= w_t + \frac{1}{2}\alpha_w \nabla_w [v_\pi(S_t) - \hat{v_{w_t}}(S_t)]^2 \\
&= w_t + \alpha_w [v_\pi(S_t) - \hat{v_{w_t}}(S_t)] \nabla_w \hat{v_{w_t}}(S_t) \\
&= w_t + \alpha_w [R_{t+1} + \gamma \hat{v_{w_t}}(S_{t+1}) - \hat{v_{w_t}}(S_t)] \nabla_w \hat{v_{w_t}}(S_t) \\
&= w_t + \alpha_w \delta_t \nabla_w \hat{v_{w_t}}(S_t)
\end{aligned}$$

Note, that we bootstrapped the true value function with the reward and next state value, as introduced in the TD learning chapter!

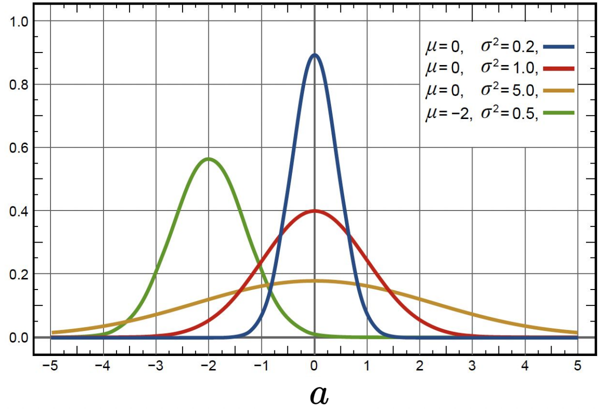
Approximating policies with gradient descent

$$\boxed{w_{t+1} = w_t + \alpha \nabla \mathcal{L}(w_t)}$$

For the policy function a parametrization with explicitly known expectation is advantageous. The natural choice is a Gaussian function. We therefore use **two function approximators** to learn the **mean and standard deviation**:

$$\pi_\theta(a|s) = \frac{1}{\sigma_\theta(s)\sqrt{2\pi}} \exp\left(-\frac{(a-\mu_\theta(s))^2}{2\sigma_\theta(s)^2}\right)$$

$$\mu_\theta(s) = y_{\theta_\mu}(s), \quad \sigma_\theta(s) = \exp\left(y_{\theta_\sigma}(s)\right)$$



We update the policy parameters to maximize the state values, using the **policy gradient theorem** to calculate the derivative:

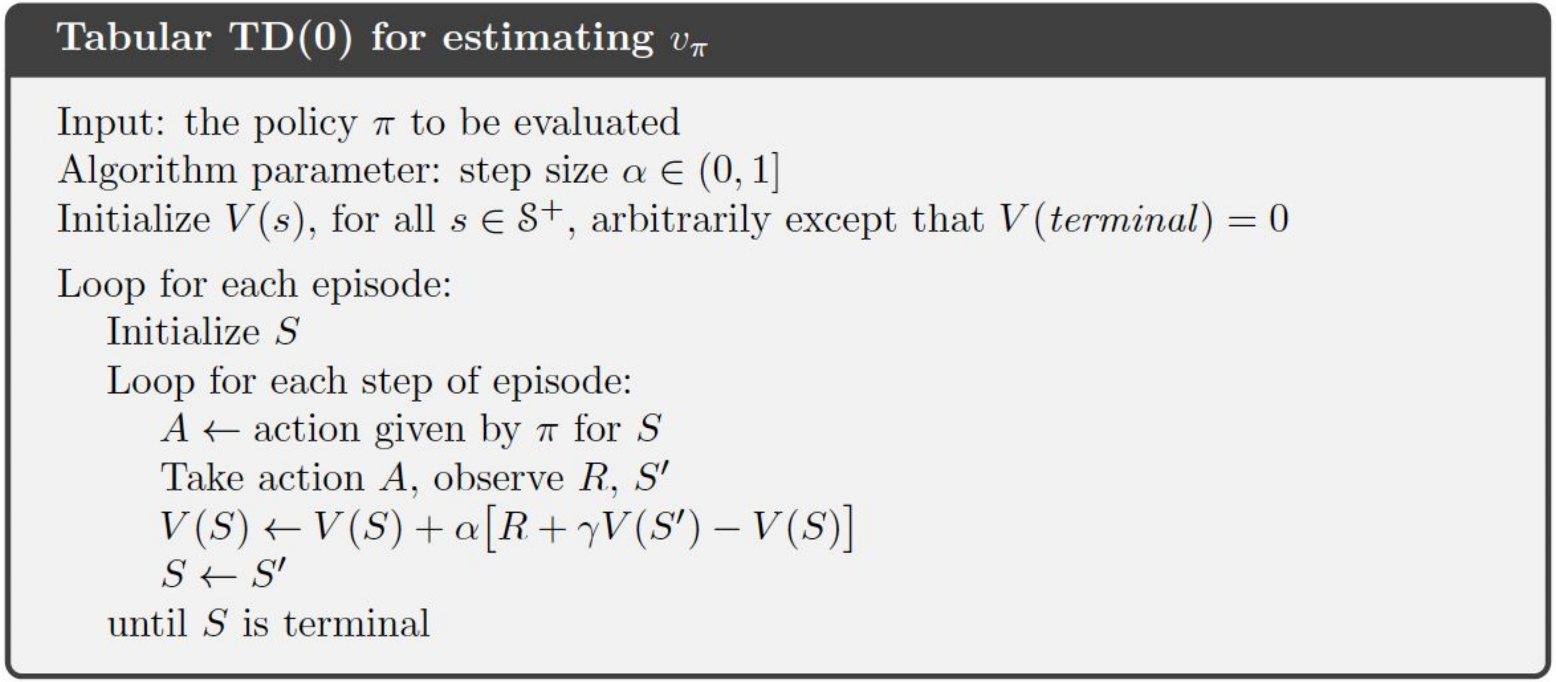
$$\nabla_\theta \hat{v_w}(s_0) \propto E_\pi[\delta_t \nabla_\theta \ln \pi_\theta(A_t|S_t)]$$

This expression is not trivial! Proof in Sutton & Barto.

⇒ update rule: $\theta_{t+1} = \theta_t + \alpha_\theta \gamma^t \delta_t \nabla_\theta \ln \pi_{\theta_t}(A_t|S_t)$

δ_t is again the TD error. γ^t is not trivial, derivation is exercise in Sutton & Barto.

TD(0) full algorithm



Tabular TD(0) for estimating v_π

```
Input: the policy $\pi$ to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\textit{terminal}) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$, observe $R$, $S'$
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

(from Sutton & Barto)

SARSA full algorithm

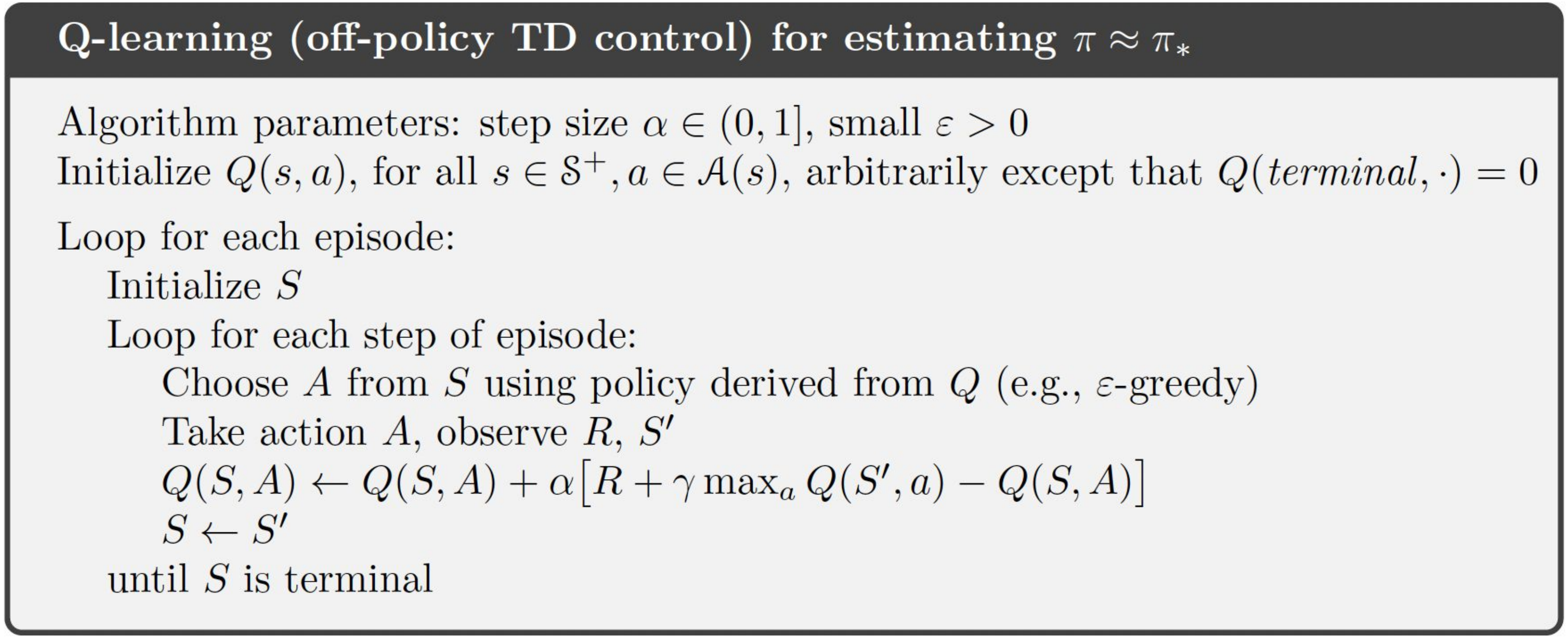
Sarsa (on-policy TD control) for estimating $Q \approx q_*$

```
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Loop for each step of episode:
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until $S$ is terminal
```

(from Sutton & Barto)

Q-learning full algorithm



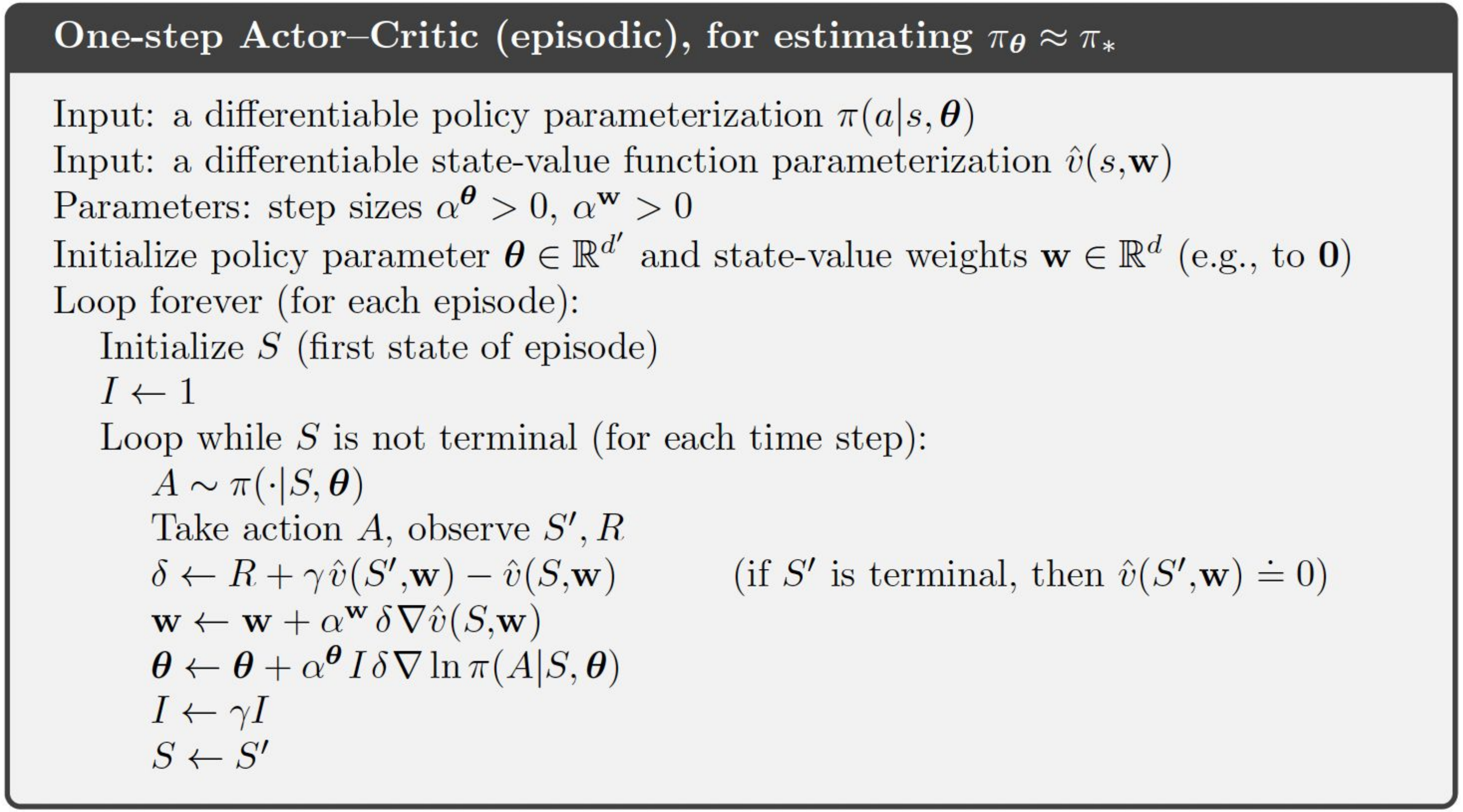
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

```
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

(from Sutton & Barto)

Actor-critic full algorithm



One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

```
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
    Initialize $S$ (first state of episode)
    $I \leftarrow 1$
    Loop while $S$ is not terminal (for each time step):
        $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
        Take action $A$, observe $S', R$
        $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$          (if $S'$ is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\,\delta\nabla\hat{v}(S,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I\delta\nabla\ln\pi(A|S,\boldsymbol{\theta})$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$
```

(from Sutton & Barto)

Additional transfer questions

How would we describe the **pendulum problem** as an MDP?

When would you prefer a **model-based** algorithm and when a **model-free** algorithm?

Tabular methods are interesting as they allow the calculation of proofs of convergence, and other mathematical properties. Can you imagine any **disadvantages** or **limitations** of tabular methods?

Can you think of any additional **challenges** when using RL with large function approximators (e.g. deep neural networks)?

Could we use different **parameterizations** of the policy function (non-Gaussian)?